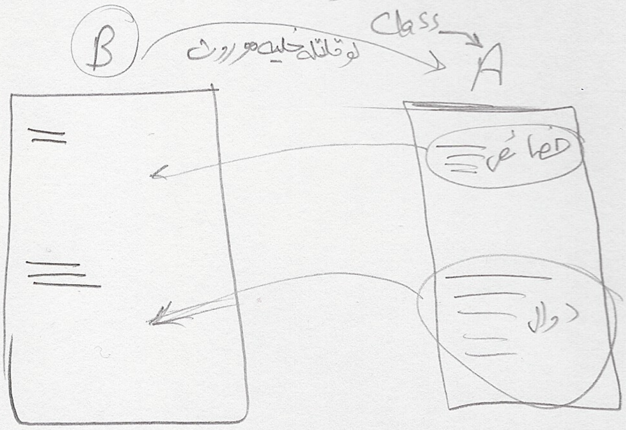
Java 102

lesson #1

inheritance = الوراثة

* باختصار لما نقول B ورث من A يعني B أخذ صفات A
* باختصار وراثة تعني " نقل الصفات " .



لو مثلاً كتبت خواص ودوال خاصة بـ B وبعدين قلتله خلي B موروث من A فذلك يعني : " انقل الخصائص الموجودة في A والدوال والـ action الموجودة في A انقلها الى الـ B " . "وفق شروط معينة"

* فرضاً أن هناك دالة موجودة في الكلاس A وتسمى Read و B موروث من A
* ثم ذهبت الى B وعرفت كائن من B

```
B Z = new B();

Z.Read();
```

← لاحظ أن Read ليست موجودة في B ولكن موجودة في A ولكن بما أن Z كائن من B و B يورث A فتعتبر Read كأنها موجودة في B ... إذاً بالإمكان استخدامها

* الوراثة توفر عليك إنك تعيد نفس الكلام وكذلك سرعة الإنتاجية
* نفترض أن B كلاس فارغ وبما أن B موروث من A فكدة أنك نقلت الأشياء الموجودة في A الى B حتى لو لا يوجد دوال
* إذاً الوراثة هي انتقال الخصائص والـ Actions الى الكلاس الجديد مع وجود أشياء إضافية

↑ بدون إعادة كتابتها

→ يعني ليس جميع الدوال والمتغيرات = درس متقدم .

Lesson 2

إنشاء كلاس

هتعمل كلاس جديد عادي
* لا جديد !!

Lesson 3

وعملية الوراثة Extends

(تم كتابة هذا الكلام عشان تقول له إن B وارث من A)

```
public class B extends A {  // أنشئ كلاس اسمه B ويرث الكلاس A

}
```

//Now :

```
B Z = new B ();
Z.printA();  // هيطبع محتويات دالة print الموجودة في A مع إن B فاضي
```

بس عشان هو وارث من A فيحق له استعمال الدوال والمتغيرات.

أوجه القوة في الوراثة

Lesson 4 :

```
public class B extends A {}
```

معنى الجملة :

* إنه هيأخذ A ويطور عليه "يوسعه عن" Extend فكأنه B هي تطوير لـ A بدون ما يعدل في A
* بإختصار : B يرث A عني أكتك بتقول أنا بطور A عني بضيف A بس بدون ما أعدل عليها

إنشاء الكلاس

Lesson 5

* من فوائد الوراثة أيضًا أو أحد أهدافها إنك لو عندك كلاس B مثلاً وعايز تضيف له أكواد عني عايز تطوره فبدل ما تعدل في نفس الملف إنت ممكن تعمل ملف جديد وتخليه يرث من هذا الملف وكده تكمل عليه عادي عشان بقا معاه نفس الصفات والدوال.

* مميزات الوراثة : * في الصيانة * عشان ترثه

وهذه مميزات قوية لبرمجة الكائنات oop.

lesson6 :

مفهوم subclass & superclass

* بما أن الـ B يرث من A إذاً نقدر نسمي A : superclass ← الأب

B : subclass ← هو الابن

* بما أن الـ C يرث من B والـ B يرث من A!

فهل إذاً الـ A يعتبر superclass للـ C ؟!

superclass

Direct superclass
∵ B extends A
∴ A : direct superclass to B
∴ B : subclass

Indirect super class
∵ C extends B & B extends A
∴ C extends A
∴ A : Indirect superclass to C
C : subclass

يعني C وارث من A عن طريق B

B : direct superclass لـ C ← وكمان

A B C D E F

F : الـ super class بتاعه هو E والبقية هي Indirect superclass

lesson 7 :-

التعرف ع كلاس object

* يوجد كلاس تلقائي في جافا يسمى object .. بمجرد أن تنشأ أي كلاس في جافا

فتلقائياً كأنك عملت B extends object بدون علمك :)

* جميع الكلاسات الموجودة في جافا موروثة من كلاس يسمى object

* يعتبر كلاس object هو الأب " لا يوجد أعلى منه "

* أعلى superclass موجود هو الكلاس object

lesson8

التعرف ع super

```
public class B extends A{
public void printB(){
    System.out.println("classB");
super.printA();
}
}
```

(A)

* تعني كلمة super : الكلاس المورث منه

* تمكنك super من الوصول إلى الأشياء الموجودة في الكلاس الأب superclass

lesson 9

Super و دالة الـ Constructor

lesson 10

* دوال الـ Constructor لا يتم توريثها بل تقوم أنت باستدعائها مع استخدام super

Ex

Class A

```
public class A {
  public int val;
  public A(int c){
    this.val = c
  }
  ......
}
```

Class B

```
public class B extends A {
  public B()
  {
    super(0);
  }
}
```

lesson 11

معلومات أكثر حول الـ Constructor

```
A x = new A();
A y = new A();
```

بعد كلمة new تأتي دالة الـ Constructor للتمهيد

x ← this.val = 5 ≡ x.val = 5

y ← this.val = 10 ≡ y.val = 10

lesson 12

استخدام أكثر من Constructor

* بالإمكان استخدام أكثر من Constructor ولكن بشرط ألا يتشابهوا في الوظيفة أو النوع "أو البارامتر".

Ex :-

```
public int val;
public A()
{ this.Val = 10; }
public A(int c)
{ this.val = c; }
public A(int a, int b)
{ this.Val = a*b;
```

~~public A(int s)~~ ← مينفعش علشان هناك Constructor آخر بنفس الوظيفة وهو public A(int c)

✓

lesson13

الـ Constructor وعملية الوراثة

B x = new B();

→ معناها: استدعي الـ Const. بتاع الـ B

* لو عندك class A و class B مثلاً:

A.Java

```
public class A {
  public int val;
  public A()
  {  This.val=10;
  }
}
```

B.Java

```
public class B extends A {
  public B()
  {  // A();   كأنك استدعيت
    this.val=500;
  }
```

يعني تسلسل التنفيذ هتكون val = 10 وبعد كدة val = 500

---

lesson14

super و Constructor

```
public class B extends A{
public B()
{
  super(70);   // كده إنت بتقول استدعي الـ Constructor بتاع الـ super (الـ A اللي هو)
               // والذي له باراميتر واحد
}
}
```

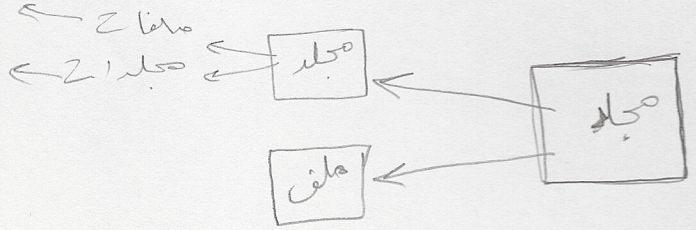
← وممكن يكون ٢ باراميتر ٣، ... أي حاجة على حسب البيانات عاملة.

Lesson 15

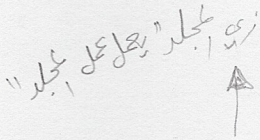
استخدام الـ package

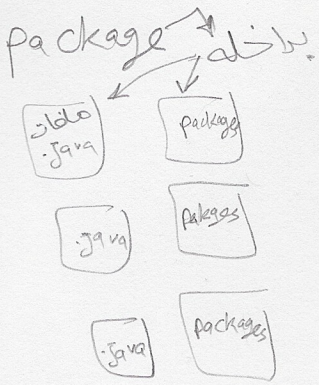
* بنا منظمة عن طريق أشياء تسمى packages .

* في نظم التشغيل



* جافا منظمة بنفس التنظيم أعلاه .





Ex: package x

بداخلها مجموعة كلاسات ومجموعة packages

ومجموعة الـ packages بداخلها أيضاً كلاسات وpackages أخرى

وهكذا

* الطريقة هذه طريقة تنظيمية فقط ما بيهمه لأسلوب المجلدات .

* تساعدنا في تنظيم الشغل والوصول لأي كلاس نريده .

* فائدة هذا: لأنه عندما نريد استخدام شيء معين لازم نحدد هو فيه موجود في أي package عشان نوصلوا ونتعامل معاه مرتقدر نستخدموا .

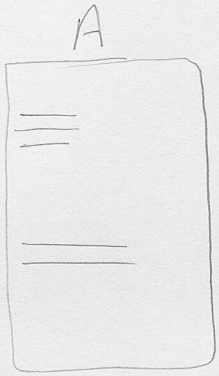
Lesson 16

مفهوم الـ Access Modifiers

public

protected

private



* إذا وضعت الـ Access Modifiers قبل المتغيرات أو الدوال فهي تحدد طريقة الوصول لهذا المتغير أو الدالة .

* إذاً باختصار الـ A.M "تتحكم في الوصول". "هل تقدر توصل للمتغير ده؟ أو الدالة دي؟"

public : وصول عام

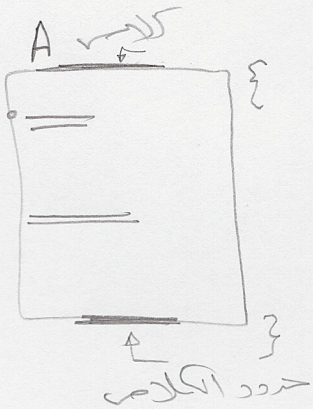
protected : وصول محمي

private : وصول خاص

Lesson 17

private حـ

private : خاص



→ private int x ;

معناه أن المتغير x لا يستخدم خارج الكلاس

فهو خاص بالكلاس ولا يستخدم خارج حدود الكلاس

Ex: A b = new A ();

b.(x) = 10 ; (XXX)

خطأ لأنه x هنا تخدم خارج حدود الكلاس

لأن الكائن قام باخراج الـ x خارج الكلاس

Lesson 18

استخدام private

```
public class A {
private int val;
public A()
{ This.val =10;
}
}
```

/* val المقصود أو هدف private وان استخدام
لن يخرج خارج القوسين */

* وظيفة private : هي الحفاظ على البيانات.

lesson 19

استخدام private مع get و set

س/ كيف نتعامل مع val - دال معمولها private - من الخارج

ج/ عن طريق setter (نرسل قيمة) و getter "عبارة عن دوال"

* يعني هي عبارة عن دوال "رجيل" إنت بتكتبها عشان ترسل أو تطلب قيم المتغير الخاص private

Ex:

```
public class A{
private int val;
  public void setVal(int c)
    {  this.val = c;
    }
  public int getVal()
    {  return this.val;
    }
}
```

set: دالة تُعين قيمة لمتغير نوعه private

get: دالة عشان تقدر تستخدم قيمة المتغير الي نوعه private

* get و set تستخدم كعُرف بين المبرمجين

---

lesson 20

فوائد استخدام private

* إخفاء البيانات

* سهولة وإضافة عمليات منطقية لأنك بتستدعيها بشكل غير مباشر

* تحكم كامل

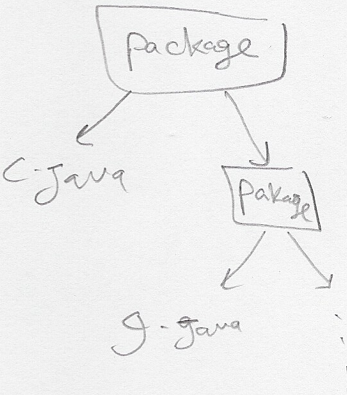
* سرعة التعديل والصيانة للكود.

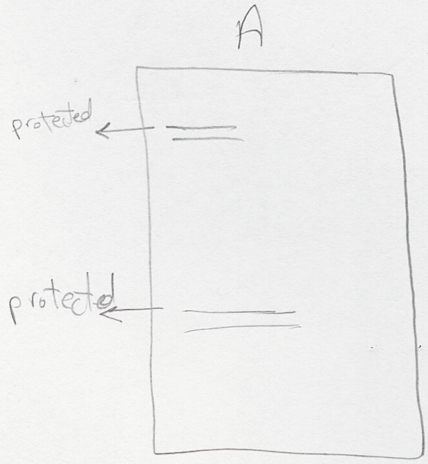
---

lesson 21

شرح protected

محمي





protected : محمي من أي كلاس أو أي مكان باستثناء حالتين :-

① أي كلاس مشتق من A ⇐ يعني أي subclass ~~مشتق من A~~

② جميع الكلاسات التي في نفس الـ package الموجود بها A

③ في نفس الكلاس المستخدم فيه

---

lesson 22

استخدام protected

شاهد الفيديو

---

lesson 23

استخدام public

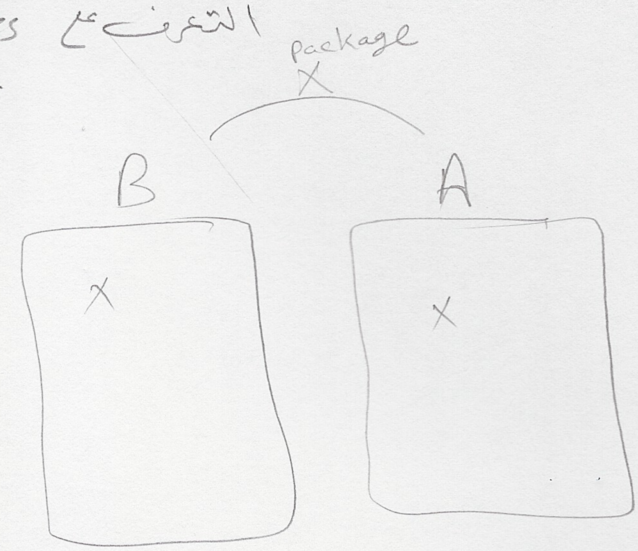
* وتعني عام وباختصار تستخدم في أي مكان

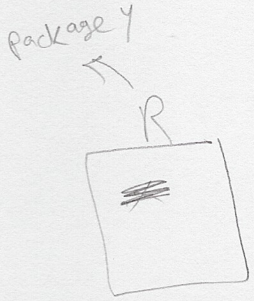
* لا يُنصح public للمتغيرات مع أنها لا تسبب أي مشكلات «حيث أنه بيتعارض مع مفهوم برمجة الكائنات»

* ينصح بجعل المتغيرات private أو protected واستخدم set و get لجلب وتخزين البيانات

lesson 24

التعرف ع Default Access Modifiers





public
protected
private

int x ؛ لم نحرر هنا

إذاً هنا يُسمى Default

⇐ وخواص هذا Access Modifier أن X مش هتقدر توصلوا إلا عن طريق الكلاسات الموجودة في نفس ال Package

⇐ يعني كلاس R لن يقدر يوصل للمتغير X

---

lesson 25

استخدام Default Access Mod.

<u>نصيحة</u>: استخدم المتغيرات مع عمل private أو protected "في الغالب" واستخدم get, set

---

lesson 26
lesson 27
lesson 28

Simple Calc.

* شاهد الفيديو

---

lesson 29
lesson 30

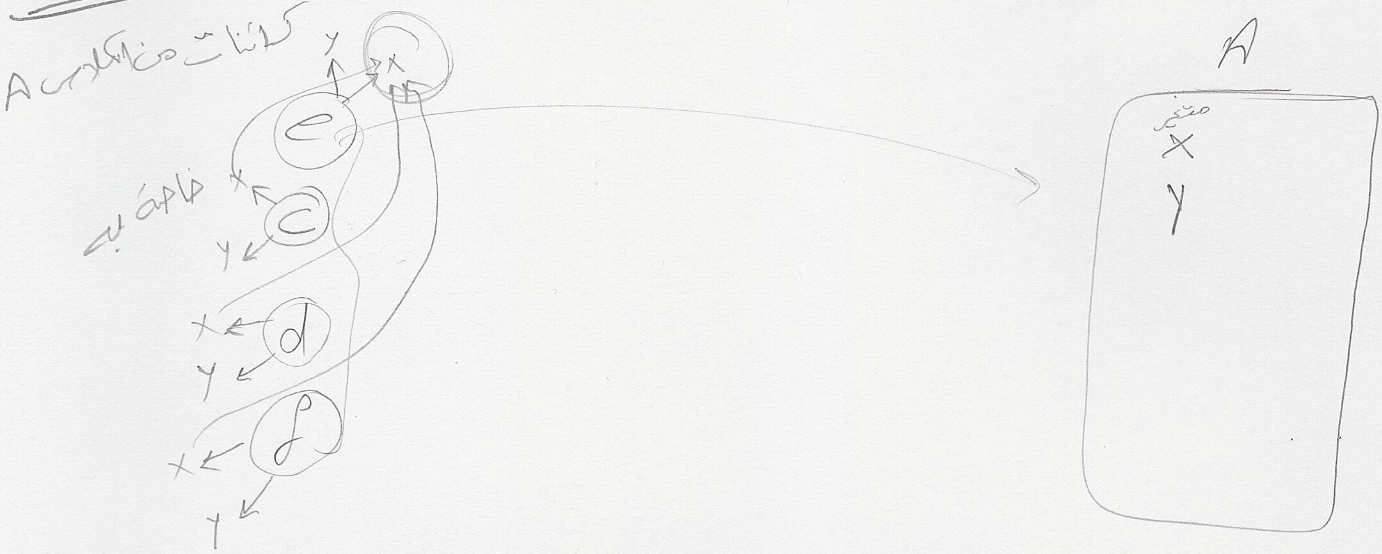
another Calc.

* شاهد الفيديو

* استخدم نوع البيانات double عشان العشرية تشتغل صح

lesson 31

التعرف على static attribute



لو كتبت عند x مثلا بعد كلمة public / private ⇐ static

ex : public static int x ;

* معناها : أن المتغير x مشترك بين جميع الكائنات التي تُنتج من الكلاس A

"ويتم تسميتها Class Member (أعضاء الكلاس) "متغير كلاس"."

* إذاً كده الـ يكتب بعده static معناه انه لا يحتاج الكائن لتعريفه أو استخدامه

يعني لما تريد الوصول لـ x وطالما أنه static فلن تحتاج كائن

* فكده أي كائن جديد له x واحدة فقط والمستخدمة في جميع الكائنات

* يعني قيمة x لو تغيرت في أي كائن ⇐ سوف تتغير في البقية.

* فأصبح الـ static هو شئ مشترك بين جميع الكائنات ولا يوجد منه إلا نسخة واحدة فقط

* للوصول للمتغير من الخارج يتم كتابته $A.X = 10;$ (A ← اسم الكلاس، X ← اسم المتغير الـ static)

* الـ Member التي من نوع static لا تحتاج لإنشاء كائن "لانها ليست خاصة بكائن معين بل مشتركة"

lesson 32

استخدام static -

```
package java102;
public class A {
    public int x; // متغير ينتج لجميع الكائنات نسخة
    public static int y; // متغير مفرد، له نسخة واحدة فقط // لجميع الكائنات //
{
```

والوصول لهذا المتغير اللي static ← يتم كتابة

```
A . y = 77 ;
```

A ← اسم الكلاس

y ← المتغير اللي نوعه static

* فائدة :- توفير تعدد الكائنات.

بواسطة إنشاء متغير يعد الكائنات التي تنتج من الكلاس A .

* أيضاً لا تحتاج إنشاء كائن للوصول للمتغير وذلك لأنه ← خاص بالكلاس وليس بالكائن

---

lesson 33

موضوعنا Static Method

```
public static void Example()
{
    System.out.println("Hi...");
}
```

* نفس الكتابة زي الدرس السابق
* هذه الدالة لا تستدعى عن طريق كائن بل عن طريق اسم الكلاس والنقطة فقط

* الدالة إذا كانت static معناها خاصة بالكلاس وليس الكائنات

Ex: A.Example();

* قاعدة هامة :- static لا يستخدم بداخلها this "السبب في الفيديو" توضح

لأنه لم يتم إنشاء كائن

* دوال الـ static تستخدم بدون إنشاء كائن.

Lesson 34

مثال على Static Method

Lesson 35

تلخيص موضوع static

```
public int x;
public static int y = 10;
```

* أي دالة أو Method أو أي إجراء procedure نوعها static ← فلا تقدر تتعامل مع المتغيرات التي بداخل الكلاس ونوعها ليس static. والعكس صحيح. "لأنه أي متغير static هيتشيخ بادئ وبيقا اي بـ static لن يتم نسخه".

* لو تم وضع private قبل static فلن تقدر تستدعيها خارج الكلاس

الا عن طريق الدالة get & set

* لا تتم إستخدام this مع static أبدًا

static يعني تقدر تستدعي بدون اشارة كائن
يعني ممكن تستدعي عن طريق اسم الكلاس

Lesson 36

مثال TheCount

```
public class TheCount {
public static int count = 0;
  public TheCount()
    { count++;          // معنى الدالة :- كل ما أنشئ كائن ← زيد الـ count بواحد
    {
}
```

Lesson 37

تابع المثال

lesson 38

مفهوم الـ Method overloading

المقصود بها : تقدر في نفس الكلاس إنك تضع دالة أخرى بنفس الإسم بإختلاف البارامترات في النوع أو العدد وممكن في الـ return type .

---

lesson 39

مثال Method overloading

شاهد video

---

lesson 40

مفهوم Method Overriding

المقصود بها : إلغي الدالة المستخدمة سابقاً في الـ superclass واستخدم مكانها الدالة الجديدة في الـ subclass .

Class A

```
public int beans(int value)
{ return 0; }
```

Class B

```
public class B extends A {
@Override
public int beans (int val)
  { return 1; }
```

* فائدتها هي تطوير الدالة بعدما ورثتها .
* تستخدم في حالة الوراثة من كلاس « superclass »

---

lesson 41

مقارنة

Method overloading	Method overriding
* إذا كتبت الـ Method واختلفت في البارامترات (بنفس الـ signiture "proto type") فتعتبر الدالة overloading	* يوجد super class به دالة اسمها مثلاً x فإذا أعدت كتابة x في الـ subclass فتسمى العملية overriding
* تستخدم في نفس الكلاس أو كلاسات مشتقة « موروثة » ، المهم يختلفوا في البارامترات عددها أو أنواعها	* كأنك بتقول إلغي الدالة الموجودة في superclass واستخدم الدالة الموجودة في الـ subclass
* لا يهتم بنوع الإرجاع	* فكرته : إعادة كتابة الدالة الموروثة
	* تستخدم في الوراثة .

lesson 42

"overloading" overCalc مثال

شاهد الفيديو

---

lesson 43

مثال ع عمل overriding للطباعة

---

lesson 44

استخدام Final

التعامل مع الثوابت

ثابت زي π مثلاً π=3.14
ده ثابت حسابي ويوجد ثوابت أخرى
ع حسب هندسة البرنامج.

* احنا عندنا متغيرات وثوابت.

مثال ع المتغير public int x=0;

مثال ع الثوابت ←

خاص
public final int y;
↳ ثابت لا تتغير قيمته

يوجد مكانين ٣ للتعامل أو وضع قيمة الثابت:

① public final int y = 10; في مكان التعريف أو أثناء الإنشاء

وكده القيمة مش هتتغير

② وضع قيمة الثابت داخل الـ Constructor أثناء تمهيد الكائن

③ وقت إنشاء الكائن عبر الـ Constructor

⇐ الـ Final يتم وضع قيمته مرة واحدة فقط وإذا تم وضعها لا تتغير

---

lesson 45

استخدام Final static

إذا كان الثابت لن تتغير قيمته بين الكائنات فيفضل بوضع Final static

باختصار حالتين:

>> إذا كان كل كائن له ثابت معين فهنا لا نضع static

>> " " كل الكائنات التي سننشأها لها نفس قيمة الثابت فيتم وضع static

لأن الثابت سيكون مشترك بين كل الكائنات. وحتى لا يتم استهلاك مساحة في الذاكرة في الفاضي.
static

Lesson 46

مثال على استخدام final static

Lesson 47

A

التعرف على Final Method

معنى كلمة final إذا جاءت مع الـ Method

```
public final int read()
{ return 0; }
```

⇐ ممنوع عمل override هذه الـ Method

يعني أي كلاس بيورث من A لازم يستخدم الدالة read كما هي بدون ما يعدل عليها override

يعني لما تحط final كأنك بتقول هذه الدالة لا تحتاج لتعديل.

⇐ استخدامها لو عايز تخلي الدالة تجمع الكلامات الموروثة ومش عايز أي كلاس يعدل عليها

التعرف على final class

Lesson 48

معنى final إذا جاءت قبل الـ class:

```
public final class A{}
```

⇐ ممنوع يُورث منه أي كلاس ومستحيل.

"يعني ما يحتاج تعديل أو تطوير"

⇐ يعني بكلام آخر الـ final تقوم بإيقاف عمل الـ extends

Lesson 49

مثال على final Method

خالد الغديري ©

lesson 50 — استخدام import

* جافا تكون محمل فيها عدد كبير من الكلاسات الجاهزة المكتوبة مسبقاً .
* يتم استدعاء هذه الكلاسات عن طريق import .
* لاستدعاء هذه الكلاسات يتم كتابة import بعد ال package " يتم الاستدعاء أولاً "

```
import java.
        applet
        awt
        beans
        ...
```

* Ex :- `import java.util.Date;` ≡ استدعي الكلاس الذي اسمه Date
كأنك كتبت مسار
  - java ← اسم ال package
  - util ← داخل java package أخرى اسمها util
  - Date ← كلاس

package java لا package util » class Date

* import تستخدم لاستيراد كلاسات جاهزة للتعامل معها .
* لاستيراد جميع الكلاسات داخل ال package يتم كتابة (*) مثال: `import java.util.*;`
ولكن لا يفضل لأنه سيكبر حجم البرنامج

---

lesson 51 — عمل import Math

* كلاس للتعامل مع العمليات الرياضية `import java.lang.Math`
* ممكن تستخدم ال package lang بدون عمل import لأنها افتراضية موجودة أصلاً .
* يعني ال system.out اللي كنا بنستخدمها قبل موجودة في lang .

---

lesson 52 — استخدام كلاس Scanner — هام

* لإدخال القيم بنفسك من المستخدم يتم استخدام كلاس scanner
* لاستدعاء هذا الكلاس `import java.util.Scanner;`

* بعدما عملت import java.util.Scanner;

* تم استدعاء الكلاس :

• Scanner f = new Scanner(system.in);

انشئ كائن جديد "الذي سيقرأ القيم" واسمه f

ويأخذ بارامتر وهذا البارامتر هو كائن تبع المدخلات "sytem.in"

* system.out ⇒ printstream ⟶ يعني كأنه يمثل الشاشة

* system.in ⇒ Input stream ⟶ يعني كأنه الشيء الذي سيدخل للشاشة

"system.in"

* in ⇒ هيمثل المدخلات عبر الشاشة

* out ⇒ المخرجات عبر الشاشة

من خواص دوال -"Method"- الكلاس Scanner

من نوع int

أنه به دالة تسمى next Int(); ⇐ تعني الانتظار للذي سيقوم المستخدم بإدخاله

Ex: Scanner f = new scanner (system.in);

system.out.println("please Enter your age:");

int c = f. next Int(); // أول ما المستخدم هيدخل القيمة سيتم وضعها داخل c

---

قراءة قيمة نصية باستخدام scanner

lesson 53

```
Scanner f = new Scanner (system.in);

String name;
system.out.println ("please enter yourname:");

name = f.next(); // string تأخذ قيمة    دالة من دوال scanner وترجعها لك

system.out.println ("your name is:" + name);
```

lesson 54

التعديل على كلاس simple calc

* يمكن تعريف الكلاس scanner كمتغير "لأنه نوع بيانات"

ex : private Scanner scn ;

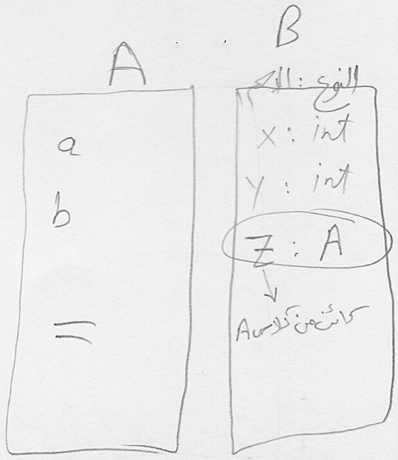
سيتم التحدث عنها بعد كده بالتفصيل

* شاهد الفيديو.

---

lesson 55

مفهوم الـ Composition



* لماذا تنشئ الكلاس ؟!
* أصلاً إنت لما تنشأ الكلاس ⇐ إذاً أنت بتنشأ نوع بيانات جديد.

نوع بيانات → Java defined

Ex : int x ; // عرفت متغير نوعه Integer

Ex : A x ; // عرفت متغير كائن نوعه A

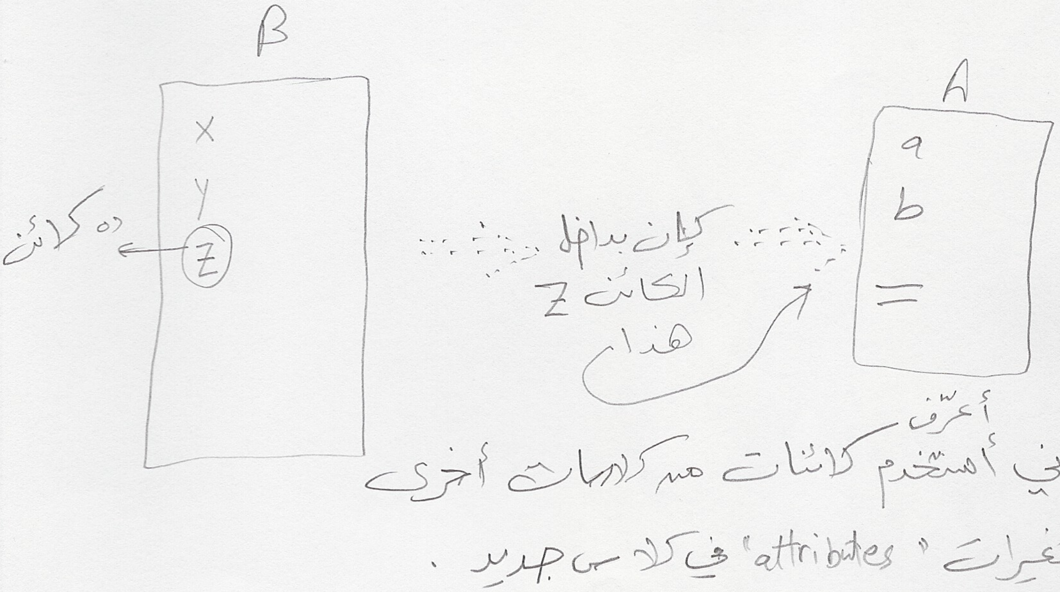
A → user defined

⇐ معنى الـ Composition : إنك تعرف كائن ⇐ هذا الكائن من كلاس آخر ليس من الأنواع الافتراضية
Ex: int, long, short...

⇐ باختصار: معنى الـ Composition : كائنات تدخل في تعريف كلاس آخر، يعني ربط.

A Z ; // كده عرفت كائن من الكلاس A

Z → ده يعتبر Attribute
→ كائن من class A



⇐ معنى الـ Composition : إني أستخدم (أعرّف) كائنات من كلاسات أخرى كمتغيرات "attributes" في كلاس جديد.

lesson 56

مثال ع مفهوم Composition

* بمجرد إنك عرفت object (من كلاس آخر) داخل الكلاس ⇐ فهذا يسمى Composition

زي إني عرفته ⇐ كده هقدر آخد خدمات الكلاس scanner // private Scanner scr ;

* إحنا لما نعرف كلاس جديد ⇐ كده إنت بتعرف نوع جديد في اللغة ⇐ عشان كده أقدر أستخدمه كخاصية .

⇐ ملخص :-

* الـ Composition : هو عبارة عن وضع كائنات من كلاسات أخرى في تعريف كلاس جديد .

* فائدته : بناء كلاسات معقدة جداً ، وسهولة التعديل .

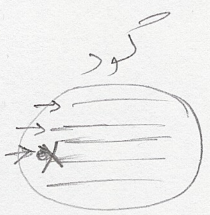
---

lesson 57 :

مقدمة للاستثناءات Exceptions

* Exceptions ≡ معالجة الأخطاء

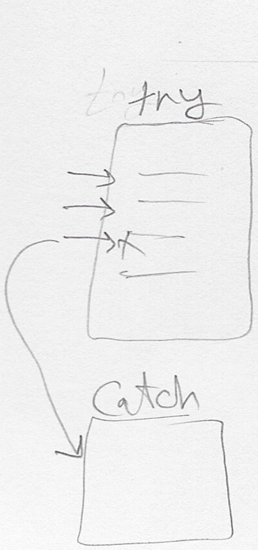
* Exceptions : هو أسلوب لمعالجة الأخطاء



وهذا الكود في احتمالية إنه يحصل له خطأ

مثال : إنت مبرمج الكود إنه يتعامل بالنت ، المستخدم فاصل النت هنا هيكون في خطأ . "Exceptions"

* Exceptions : يعني غير العملية السير الكود "غير التحكم بالكود"



⇐ افرض عندي كود ومحتمل إنه يقع فيه مشكلة

كده لازم تحط الكود داخل بلوك اسمه try يعني جرب هذا الكود

⇐ إنت لما حطيت الكود داخل بلوك try كده إنت حطيت كود مشكوك في أمره يعني ربما يحدث إستثناء "خطأ"

⇐ Catch : إذا حدث خطأ في بلوك try هيعمل jump ويروح لـ catch

و لو مفيش خطأ : خلاص مش هيروح لـ catch

* يعني في البلوك Catch إنت هتقله إزاي يعالج الخطأ

---

lesson 58 :-

مثال ع الاستثناءات

---

lesson 59

استخدام try & catch

* يتم وضع الكود المشبوه داخل try { }
يعني اللي فيه احتمالية خطأ

* يجب استخدام catch أينما استخدمت try

```
try {
   ...
}
catch( Exception e)
{
   --
}
```

(Exception ← كلاس الخطأ معالجة ، e ← كائن)

* ميزة ال Exception إنها تعطيك فرصة قبل إنهاء البرنامج
يعني قبل إيقاف البرنامج "يعني لو مفيش إستثناء البرنامج هيقف"

---

lesson 60

الكائن المستخدم مع Catch

```
try {
  →
  → X →→ عملية رمي Throw
}
catch( Exception e)
{

}
```

(e ← نوعه Exception ← عرف كائن اسمه e)

* عندما يحصل خطأ ← ترسل بيانات الأخطاء بشكل كائن يسمى Exception الذي تمت بتعريفه بين قوسين Catch

* المعلومات الخاصة بالخطأ الذي وقع يتم تخزينها بالكائن الذي انشئته من الكلاس Exception الي هو e

lesson 61

مثال لإستخدام كائن catch

* شاهد الفيديو

* Exception هي استثناء عام " جميع كلاسات الاستثناءات مشتقة منه ".

* يفيد هذا السطر في معرفة نوع الاستثناء : Java.util.InputMismatchException output

هذا اسم كلاس الاستثناء

* يعني لو جيت غيرت نوع الكلاس في التعريف

```
catch(InputMismatchException e)
{
  ---
  ---
  ---
}
```

ولكن هذا الكود داخل ال catch لن يتم تنفيذه إلا إذا حدث خطأ من النوع InputMismatchException

---

lesson 62

استخدام أكثر من catch

ببساطة أضف كود catch مرة أخرى

```
catch(){...}
catch(){ }
```

* يمكنك وضع عدد لا محدود من ال catch على حسب احتمالات الأخطاء التي عندك.

---

lesson 63

استخدام catch واحدة للجميع

* وذلك باستخدام الكلاس Exception لأن باقي الكلاسات بتاعت الاستثناءات مشتقة منه

```
try {...}

catch(Exception e) {...}
```

Lesson 64

استخدام empty Catch

* معناها: "لا أريد أن أتخذ أي إجراء ⇐ وكمل البرنامج عادي "مع انه فيه خطأ""

* ال Catch الفارغ هو عبارة عن عدم إتخاذ أي إجراء مع الخطأ.

Lesson 65

التعرف مع Finally

```
try { ...
}
Catch(Exception e) {
  ...
}
Finally { ...
  ...
}
```

* Finally ⇐ يأتي بآخر ال Exception

* في وجود استثناء أو عدم وجود استثناء ستنفذ ال Finally

* فائدته ⇐ لتنظيف الذاكرة / يعني الموارد التي استخدمتها
⇐ تنظيف البرنامج من المصادر التي استخدمها ولم يستفيد منها

* فكرته: إنقاذ ما يمكن إنقاذه

Lesson 66

التعامل مع Finally و Return

```
public static void main(String[] arg) {
  . openNet....
  .
  .
  .
  . Catch(Exception e) {
      return;      // main يعني اخرج من
  }
  Finally {
    ......
  }
  System.out.....
}
```

Return; "اخرج من الدالة"

بس كده الحاجات لسه مفتوحة
يعني مستخدمة "يعني لو حاجز مكان بالذاكرة هيكون لسه محجوز"

عشان كده دي بيوضح قوة استخدام Finally

لن ينفذ هذا الكود - يعني هينفذ Finally وثم يعمل Return
ولو حطيت Return هينفذه.

lesson 67 — التعرف مع throw

* بسبب أن الدالة nextInt() ترمي استثناء لذلك وضعناها داخل بلوك try يعني شاء scanner كلاس معرف من قبل جافا .

* يعني لازم تعمل دالة ← ترمي استثناء

```
package Java 102

public class A {
  public void fun(int val)
    if (val > 100)
    {
      throw new Exception ("Value over flow");
    }
```

- throw ← يعني ارمي استثناء
- new Exception ← كده بعرف كائن استثناءات
- الكود ناقص بستابع //

* كلمة throw لازم يأتي بعدها كائن "الكائن الذي سيرمى"

lesson68:-

التعرف مع Exception catcher

* أي دالة ترمي استثناء تسمى Exception thrower
* وأي دالة ترمي استثناء يجب كتابتها (أثناء استدعائها) داخل بلوك try

"أنواع دوال الاستثناءات" Exception thrower

- exception catcher
- exception propagator ← الدرس القادم

exception catcher

معناه: الدالة ترمي الاستثناء وتعالجه بنفسها
يعني throw تكون بداخل بلوك try

```
try { If (val > 100)
    { throw new Exception ("Error");
    } }
catch (Exception e) { }
```

e ← هيستقبل الاستثناء الذي رمي من هنا (throw)

lesson 69

التعرف ع Exception propagator

```
public void fun(int val) throws Exception
{
    if(val > 100)
    {
        throw new Exception("Error");
    }
}
```

(اسم الكلاس)

* Exception propagator : ترمي (الدالة) استثناء ولا تعالجه
* يعني تحتاج إنت تعالجه في أي مكان آخر وليس لازم في نفس الدالة
* يسمى propagator لأن الاستثناء خرج من الدالة وانتشر وراح لكود آخر "الي هو مكان الاستدعاء" ← والي هو مسؤول انه يعمله catch ← انت الي هتكتبه ...
* propagate : يعني الاستثناء رمي ولكن لم يعولج !! (← تم معالجته)
* تم المعالجة عن طريق لما تستدعي الدالة هتحط الدالة في بلوك try و catch في ال main .

يعني هذه الدالة ترمي استثناء ونوعه propagator

وبالإمكان كتابة أكثر من نوع ولكن يجب عمل import للكلاس لأن كلاس ال Exception افتراضي ولا يحتاج عمل import

Ex : public void fun(int val) throws Exception, InputMismatchException

يعني دالة fun هترمي واحد من الاستثناءات التالية

يعني لما تيجي تعمل catch ⇐ هتعمله لأي كلاس من المكتوبين

---

lesson 70

مثال ع Exception Catcher

---

lesson 71

مثال ع Exception propagator

Lesson 72

propagator و Runtime Exception نقطة مهمة جدا

* Runtime Exception: استثناء لا يحدث إلا وقت التشغيل مثل InputMismatchException

لأن الـ input تحتاج تشغيل لتأخذ المدخلات

* Runtime Exception: لا تقدر تتعرف فيه إلا بعد تشغيل البرنامج

```
import InputMismatchException ;
public class A {
    public void setMaxValue(int val) throws InputMismatchException
    { if (val > 100)
        throw new InputMismatchException ("Error");
      System.out.println(val);
    }
}
```

لو شلت هذا السطر مش هيحصل Error "يعني مش هيحصل Exception"

* إذا كان نوع الـ Exception هو Runtime ⇐ دالة الـ propagator يكون موقع السطر هذا اختياري.

* يعني لما تلاقي كده فاعرف إن الاستثناء ده يحصل وقت التشغيل Runtime.

Lesson 73

Checked & unchecked Exceptions

Exceptions: "أنواع الاستثناءات نفسها"

Checked

* إذا كان الاستثناء يُشتبك عليه أثناء الـ Compile time
  * وقت تشغيل البرنامج
  * أو أثناء كتابة البرنامج
* يعني يتأكد من الاستثناء قبل التشغيل
* يكشف المشاكل قبل تشغيل البرنامج
* يفحص قبل التشغيل

un checked

* يحدث Runtime Exception
* يعني هذا النوع لن يتم التشيك عليه إلا وقت التنفيذ لأن أنا لا أعرف هل هذا استثناء أم لم يحدث
* مثال: القسمة على صفر. "هذا استثناء لم نشيك عليه وقت الـ Compile"
* يكشف المشاكل بعد تشغيل البرنامج
* يفحص بعد التشغيل

lesson 74 — أنواع كلاسات الـ Exception

www.docs.oracle.com/javase/7/docs/api/index.html

Lesson 75 — التعليق مع java.lang

* java.lang.* anything : يعني لو استخدمت أي كلاس منها ⇐ مايحتاج إنك تعمل import

لأن دي حاجات افتراضية في اللغة " language "

* java.lang : عبارة عن package افتراضية .

Lesson 76 — إنشاء كلاس MyException

* أول حاجة اعمل كلاس جديد عادي
* بعد كده خلي الكلاس ده يرث من كلاس Exception ⇐ extends Exception
- الكلاس MyException الكونستركتور بتاعه بياخد رسالة نصية MyException("Error")

Lesson 77 — تابع المثال

---

lesson 78 — التعرف على assertion

* assert (تأكد) تستخدم في حالة أي Expression ، وهذا الـ Expression يعطي ⇐ true / false
* Expression زي مثلا لو X > Y ... الخ .
* نتيجة الـ Expression لازم تكون true أو false ، لو true يقام ، ولو false هيعمل استثناء AssertionError
* فكرة assert هي التعامل مع الأخطاء المنطقية Logical Errors

Ex:
```
int x, y;
x = 10;
y = 5;
assert x > y;   // تأكد أن x أكبر من y
```
(المعنى)

* لو حدث خطأ " التأكد طلع غلط " سيتم رمي استثناء يسمى AssertionError
* يجب تفعيل الـ assert لأنها تحتاج تفعيل (يعني مش بتشتغل تلقائيا) ⇐ -ea

Run » Set project Config. » Customize » vm (virtual machine) » -ea

lesson 79

التعرف ع صيغة أخرى لـ assert

الرسالة اللي عايزها

```
assert x>y : "x less than y";
```

⟵ يعني لما يحدث خطأ استثناء هيظهر وصف للخطأ مكتوب فيه x less than y.

---

lesson 80

مقارنة بين assert و If

* بالإمكان تقريباً استعمالها ومعاملتها If ولكن للأخطاء
* يعني في المثال لو لم يتحقق الشرط سيعطي خطأ "Error" "استثناء"

بينما If مفيهاش أخطاء

```
ex : assert ((x<=10) && (y<=100)) : "Error";
```

إذا لم يتحقق الشرط هيظهر الرسالة

---

lesson 81

مفهوم الـ polymorphism

هام جداً جداً جداً جداً

* ترجمتها : تعدد الأشكال

lesson 82

* polymorphism ⟵ يعني تستدعي الدالة بدون معرفة الكائن الذي ستستدعي منه
⟵ يعني هتستدعي دالة والكائن مجهول

```
public void poly (A value)
{
    value.print();
}
```

value ⟵ كائن من الكلاس A أو أحد مشتقاته

* الدالة واحدة لكن الذي يتغير هو الكائنات

lesson 83

* الدالة تقدر تنفذها بأكثر من شكل ع حسب الكائن الذي ستكتبه في البارامتر وقت التنفيذ
* معنى آخر لتعدد الأشكال ⟵ الدالة زي ما هيا والذي بيتغير هو الكائن وبالتالي ستتغير طريقة التنفيذ بناءً ع الكود الموجود في الكائنات المختلفة المستخدمة

lesson 84

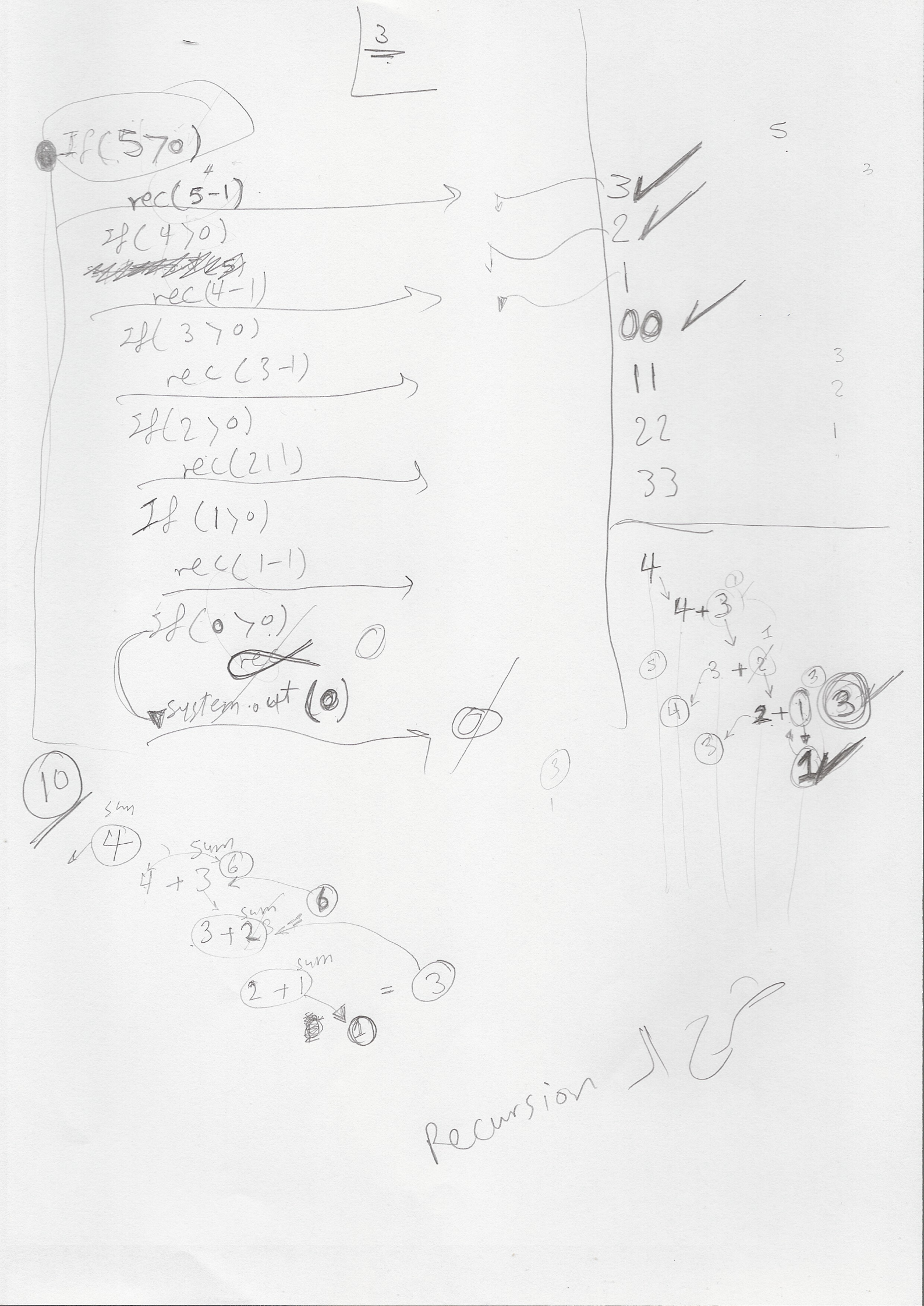
مثال

lesson 85

* في الـ polymorphism الكلاس الأساسي لازم يوفر الدالة ع أساس الكلاسات الأخرى تعمل لها override بما يوافقها
* إذا أنشأت دالة جديدة ومررت الكلاس الأب ⟵ تقدر تطبق عليها حركة الـ polymorphism

lesson 86:

lesson 87:


3
If(5>0)
rec(5-1)
If(4>0)
rec(4-1)
If(3>0)
rec(3-1)
If(2>0)
rec(2-1)
If(1>0)
rec(1-1)
If(0>0)
System.out(0)
3
2
1
00
11
22
33
4
4+3
3+2
2+1
10
sum
4+3=6
3+2=5
2+1=3
Recursion

lesson 88

شرح Recursive Method

⇐ هامة راجعها كويس

⇐ هي عبارة عن دالة تستدعي نفسها ⇐ يعني بداخل نفس التعريف يوجد استدعاء لنفس الدالة

⇐ يجب وضع حد لدالة ال Recursive حتى لا يحدث خطأ.

---

lesson 89

مثال

فائدتها: * التعامل مع ال tree

* " " البيانات المتداخلة

* محاولة تكرار عمليات

* المضروب ! .... الخ

* شاهد الفيديو

---

lesson 90

كلاس Student

* init: نوع بيانات معرف من قبل مسبقاً ويحتفظ بقيمة واحدة فقط.

lesson 91

* الكلاس يكون بداخله مجموعة بيانات مغلفة داخل الكائن المعرف من قبل المستخدم
"Incapsulation" التغليف

lesson 92

* كتابة الدوال هكذا: setName
small · Capital

lesson 93

يسمى Camel style ستايل الجمل

* يوجد ٣ أنواع من الستايلات:

* C - style ⟵ مثال set_name() كل الكلمات صغيرة بينها _

* pascal - style ⟵ SetName() كل الكلمات Capital

* Camel style ⟵ setName
small · Capital

lesson 94
" 95
" 96
" 97
" 98
" 99

lesson 100 كلاس Tweet

Lesson 101

// 102

// 103

// 104

---

lesson 105 كلاس Calculator

lesson 106

---

lesson 107 عمل واجهة باطاطا

```
import javax.swing.JFram;
```

---

lesson 108 كلاس operating system

// 109

// 110

// 111

// 112

// 113

---

lesson 114 نقطة مهمة جدا موضوع polymorphism

---

lesson 115 ختام الدورة 102

^^
